

DEVELOPMENT AND PERFORMANCE OF *CAMEL_AERO*, A TRULY MATRIX-FREE, PARALLEL AND VECTORIZED UNSTRUCTURED FINITE VOLUME SOLVER FOR COMPRESSIBLE FLOWS

Shuangzhang Tu^{*}, Marvin Watts, Andrew Fuller, Reena Patel, and Shahrouz Aliabadi

Northrop Grumman Center at Jackson State University, Jackson, MS 39204, USA

ABSTRACT

This paper reports the development and performance of *CaMEL_Aero*, our truly matrix-free, parallel and vectorized unstructured finite volume solver for compressible flows. The Jacobian-free GMRES method is used to solve the linear systems of equations inside each nonlinear Newton-Raphson iteration. Furthermore, the matrix-free Lower-Upper Symmetric Gauss Seidel (LU-SGS) method is employed as a preconditioning technique to the GMRES solver. The solver is parallelized using mesh partitioning and Message Passing Interface (MPI) functions. The solver is also vectorized using two main vectorization techniques: the face coloring algorithm to vectorize the long loops over faces and the truncated Neumann expansions of the inverse of preconditioning matrices to vectorize the LU-SGS preconditioner, respectively. A few 2D and 3D numerical examples are presented to demonstrate the performance of the present solver.

1. INTRODUCTION

We have developed a cell-centered finite volume (FV) solver named *CaMEL_Aero* for high-speed compressible flows. The goal of this solver is to solve practical aerodynamic problems arising from aerospace and automotive industries in an accurate, efficient and robust way.

In *CaMEL_Aero*, implicit time integration is adopted to obtain better efficiency, especially for high Reynolds number flows. The implicit time integration results in a large nonlinear system of equations for complex 3D applications. The Newton-Raphson iterative method is used to solve this nonlinear system. Inside each Newton-Raphson nonlinear iteration, a large, sparse and usually ill-conditioned linear system must be solved. The Generalized Minimal RESidual (GMRES) solver (Saad 1996) has been widely used in solving large sparse linear systems. The GMRES solver involves only matrix-vector multiplication, thus a Jacobian-free (Knoll and Keyes 2004) implementation is possible. Like other iterative methods, the performance of the GMRES solver is highly

related to the preconditioning. Though the GMRES solver itself can be matrix-free (i.e. Jacobian-free), the preconditioning technique is usually not matrix-free. A matrix-free preconditioning approach was introduced (Luo, Baum et al. 1998) for the GMRES solver. In that approach, the Jacobian obtained from the low-order dissipative flux function is used to precondition the Jacobian matrix obtained from higher-order flux functions. Using the approximate Lower Upper-Symmetric Gauss-Seidel (LU-SGS) factorization of the preconditioning matrix, their preconditioning is truly matrix-free. We combine the Jacobian-free GMRES solver and the matrix-free LU-SGS solver in *CaMEL_Aero* in a massively distributed memory environment.

Parallel computing becomes indispensable for large-scale simulations. If the parallel computer is also equipped with multi-streaming and vector processors, the process of simulating large-scale problems will be further accelerated. The Cray X1E is such a supercomputer built using a hybrid parallel, vector, and multi-streaming design. Codes running on the Cray X1E must be fully vectorized for best performance. In *CaMEL_Aero*, two main subroutines need to be vectorized. Both subroutines are called extensively by the Jacobian-free GMRES solver and consume the vast majority of the total computational time. One subroutine is about a long loop over faces. In our approach, we separate all faces into groups according to the so-called “face coloring” algorithm (Tu, Aliabadi et al. 2005b). Another subroutine is about the LU-SGS preconditioning. Because this preconditioning is based on the LU approximate factorization, to avoid the sequential computations in solving the triangular system and make vectorization possible, we apply the approximate truncated Neumann expansions of the inverse of the triangular matrices (Benzi and Tuma 1999).

The purpose of this paper is to report the development and performance of *CaMEL_Aero*. In Section 2, we will briefly review some key ingredients of the development of *CaMEL_Aero*. Section 3 provides a detailed description of the vectorization techniques. In Section 4, we will present several 2D and 3D numerical examples to demonstrate the performance of the solver.

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 01 NOV 2006		2. REPORT TYPE N/A		3. DATES COVERED -	
4. TITLE AND SUBTITLE Development And Performance Of Camel_Aero, A Truly Matrix-Free, Parallel And Vectorized Unstructured Finite Volume Solver For Compressible Flows				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Northrop Grumman Center at Jackson State University, Jackson, MS 39204, USA				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited					
13. SUPPLEMENTARY NOTES See also ADM002075., The original document contains color images.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 8	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

2. NUMERICAL METHOD

Following the standard finite volume discretization (hybrid tetrahedral, pyramidal, prismatic and hexahedral cells allowed) and using the implicit backward Euler formula (BDF) for the time integration, we can obtain the following discrete form of the compressible Navier-Stokes equations for each cell i

$$\frac{\alpha_i \mathbf{U}^{n+1} + \alpha_0 \mathbf{U}^n + \alpha_{-1} \mathbf{U}^{n-1}}{\Delta t} + \mathbf{R}(\mathbf{U}^{n+1}) = 0 \quad (1)$$

with

$$\mathbf{R}(\mathbf{U}) = \frac{1}{|\Omega_i|} \sum_{k=1}^{n_f} (\mathbf{H} \cdot \mathbf{n} \Delta s)_k \quad (2)$$

Here \mathbf{H} includes both inviscid and viscous fluxes, \mathbf{n} is the outward unit normal vector of the faces surrounding cell i and $|\Omega_i|$ is the volume of cell i . n_f is the number of faces of the cell and Δs is the area of k th face of cell i . $\alpha_i = 1.0$, $\alpha_0 = -1.0$ and $\alpha_{-1} = 0.0$ for first order time accurate scheme (BDF1). $\alpha_i = 1.5$, $\alpha_0 = -2.0$ and $\alpha_{-1} = 0.5$ for second order time accurate scheme (BDF2).

Remarks.

- The spatial accuracy of the present solver is second-order based on linear reconstruction.
- The inviscid flux across the interface is computed through the HLLC (Harten, Lax et al. 1983; Toro 1999) flux function or the modified Steger-Warming (Scalabrin and Boyd 2005) flux function.
- The solution gradient across the cell interface is computed via the directional derivative method for viscous fluxes (Mathur and Murphy 1997).
- The one-equation Spalart-Allmaras Detached Eddy Simulation (SA-DES) (Nikitin, Nicoud et al. 2000) turbulence model is used to compute the turbulent eddy viscosity.

2.1 Jacobian-free GMRES solver

We can use $\mathbf{G}(\mathbf{U})$ to stand for the left hand side of Eq. (1), i.e. $\mathbf{G}(\mathbf{U}) = 0$. The standard Newton-Raphson iterative method is used to solve this non-linear system, leading to

$$\mathbf{J} \delta \mathbf{U} = -\mathbf{G}(\mathbf{U}) \quad (3)$$

where \mathbf{J} is the Jacobian matrix and can be computed via

$$\mathbf{J} \equiv \frac{\partial \mathbf{G}}{\partial \mathbf{U}} = \frac{\alpha_i}{\Delta t} \mathbf{I} + \frac{\partial \mathbf{R}}{\partial \mathbf{U}} = \frac{\alpha_i}{\Delta t} \mathbf{I} + \tilde{\mathbf{J}} \quad (4)$$

where $\tilde{\mathbf{J}}$ denotes the contribution to \mathbf{J} from the spatial flux terms and \mathbf{I} is the identity matrix.

Inside each non-linear Newton-Raphson iteration, a linear system described as Eq. (3) must be solved. This usually huge, sparse and ill-conditioned linear system is solved by the Generalized Minimal RESidual method (GMRES) (Saad 1996). Because the Krylov space based algorithm involves only matrix-vector multiplication, it is unnecessary to form the Jacobian matrix explicitly. We are able to approximate the matrix-vector product using (Knoll and Keyes 2004)

$$\tilde{\mathbf{J}} \mathbf{v} \approx [\mathbf{R}(\mathbf{U} + \varepsilon \mathbf{v}) - \mathbf{R}(\mathbf{U})] / \varepsilon \quad (5)$$

where \mathbf{R} is evaluated according to Eq. (2). Only the spatial contribution $\tilde{\mathbf{J}}$ in Eq. (4) needs this approximation because the time-dependent term can be evaluated exactly. In Eq. (5), the choice of ε is a balance between the approximation accuracy and the floating point rounding error. This approximation has the following advantages: (i) avoid the difficulty and cost in forming the Jacobian matrix; and (ii) save a significant amount of memory for storing the Jacobian matrix.

2.2 Matrix-free LU-SGS preconditioning

In *CaMEL_Aero*, we adopt the Lower-Upper Symmetric Gauss Seidel (LU-SGS) method (Luo, Baum et al. 1998) as a preconditioning technique. The Jacobian matrix of the low-order dissipative flux function can be trivially obtained and is more diagonally dominant and more compact than the high-order Jacobian matrix. Therefore, the low-order Jacobian matrix is a good candidate as preconditioner to the high-order Jacobian matrix. We use the simplest and most dissipative flux function, the local Lax-Friedrich (LF) flux to establish the preconditioning matrix. The local LF flux normal to the cell interface can be expressed as

$$\mathbf{F}_{LF} = \mathbf{T}^{-1} \left\{ \frac{1}{2} [\mathbf{F}(\mathbf{T}\mathbf{U}_i) + \mathbf{F}(\mathbf{T}\mathbf{U}_j) - \lambda^* (\mathbf{T}\mathbf{U}_j - \mathbf{T}\mathbf{U}_i)] \right\} \quad (6)$$

where i and j are the indices of the left and right adjacent cells of the face, respectively, \mathbf{T} is the orthonormal rotation matrix of the face, and λ^* represents the largest wave speed in the direction normal to the interface (Tu, Aliabadi et al. 2005a).

The Jacobian matrix of the flux function described as Eq. (6) can be conveniently separated into block diagonal

part, lower block triangular part and upper block triangular part (Luo, Baum et al. 1998; Sharov, Luo et al. 2000), i.e.

$$\mathbf{J}_{low} = \mathbf{D} + \mathbf{L} + \mathbf{U} \quad (7)$$

where the subscript ‘low’ indicates that the Jacobian matrix comes from the low-order dissipative flux function. Assuming that $j < i$ in Eq. (6), we obtain the \mathbf{L} operator for cell i contributed by cell j

$$\mathbf{L}_{ij} = \frac{1}{2|\Omega_i|} \mathbf{T}^{-1} \left[\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U}_j)}{\partial (\mathbf{T}\mathbf{U}_j)} - \lambda^* \mathbf{I} \right] \Delta s \mathbf{T} \quad (8)$$

If $j > i$ in Eq. (6), then the \mathbf{U} operator is obtained. The diagonal block for row i of \mathbf{J}_{low} can be expressed as

$$\mathbf{D}_i = \left(\frac{\alpha_1}{\Delta t} + \frac{1}{2|\Omega_i|} \sum_{k=1}^{n_f} \lambda_k^* \Delta s_k \right) \mathbf{I} \quad (9)$$

which can be represented by a single scalar for each cell. Note that the time dependent term is included in \mathbf{D} .

The preconditioning matrix is taken as the approximate Lower Upper-Symmetric Gauss-Seidel (LU-SGS) factorization of \mathbf{J}_{low} , namely,

$$\mathbf{P} = (\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}) \quad (10)$$

By applying the right-preconditioning to the Jacobian-free GMRES solver, we obtain the new form of Eq. (5).

$$\tilde{\mathbf{J}}\mathbf{P}^{-1}\mathbf{v} \approx [\mathbf{R}(\mathbf{U} + \varepsilon\mathbf{P}^{-1}\mathbf{v}) - \mathbf{R}(\mathbf{U})]/\varepsilon \quad (11)$$

It has to be stressed that the function \mathbf{R} in Eqs. (2), (5) and (11) is evaluated following the second order reconstruction procedure. Before Eq. (11) can be implemented, $\mathbf{v}^\# = \mathbf{P}^{-1}\mathbf{v}$ must be solved first. This can be done by solving

$$\mathbf{P}\mathbf{v}^\# = \mathbf{v} \quad (12)$$

for $\mathbf{v}^\#$. Substituting Eq. (10) into Eq. (12) yields

$$(\mathbf{D} + \mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U})\mathbf{v}^\# = \mathbf{v} \quad (13)$$

which can be solved in two steps in which the block forward sweep

$$(\mathbf{D} + \mathbf{L})\mathbf{v}^* = \mathbf{v} \quad (14)$$

is followed by the block backward sweep

$$(\mathbf{D} + \mathbf{U})\mathbf{v}^\# = \mathbf{D}\mathbf{v}^*. \quad (15)$$

In Eqs. (14) and (15), the \mathbf{L} and \mathbf{U} operators are computed when needed, thus completely eliminating the need to store the preconditioning matrix. In the parallel version, the LU-SGS preconditioning is implemented locally on each processor to avoid inter-processor communications. Numerical experience shows that this approximation still yields satisfactory convergence performance.

2.3 A simple slope limiting procedure

The linear reconstruction of a component of the primitive vector \mathbf{q} , denoted by q , can be expressed for cell 0 as

$$q_k^R = q_0 + \phi \Delta \mathbf{r}_k^T \nabla q_0 \quad (16)$$

where q_k^R is the reconstructed solution at the center of the k th face of cell 0, q_0 is the solution at the cell center, $\Delta \mathbf{r}_k$ is the distance vector between the face center and the cell center, ∇q_0 is the unlimited gradient vector that is computed according to the Gauss theorem and ϕ is the slope limiting factor.

The slope limiter is employed to suppress unphysical overshoots/undershoots. We have employed an effective slope limiter for triangular and tetrahedral meshes in (Tu and Aliabadi 2005a; Tu, Aliabadi et al. 2005c). We found that simple extension of that limiter to other types of meshes will introduce excessive dissipation. Therefore, we design a new limiter that is simple, effective and suitable for any types of cells. The limiting procedure ensures that no new extrema are allowed during reconstruction.

For compressible flows, density ρ is used to compute the slope limiter. All components of the primitive vector \mathbf{q} use this same limiter. We first compute the allowable density variation in each cell via

$$\begin{aligned} \Delta \rho_{\max}^c &= \max(\rho_{\max}^c - \rho_o, \varepsilon \rho_o) \\ \Delta \rho_{\min}^c &= \min(\rho_{\min}^c - \rho_o, -\varepsilon \rho_o) \end{aligned} \quad (17)$$

where ρ_{\max}^c and ρ_{\min}^c are the maximum and minimum density around the cell, respectively. The beauty of the above formulation is that they allow some small amount

of numerical noise by adjusting positive parameter ε . This is important because we do not want the limiter to be active in regions where the solution is smooth with negligible numerical noise. Numerical experiences show that $\varepsilon = 10^{-3} - 10^{-4}$ is sufficient to suppress unphysical overshoots/undershoots while not affecting the residual convergence too much. Also, note that the computed $\Delta\rho_{\max}^c$ and $\Delta\rho_{\min}^c$ are always positive and negative, respectively. We then calculate the unlimited variation of density at each vertex i of cell θ using the unlimited density gradient $\nabla\rho_0$.

$$\Delta\rho_i^v = \Delta\mathbf{r}_i^T \nabla\rho_0 \quad (18)$$

We compare each $\Delta\rho_i^v$ with $\Delta\rho_{\max}^c$ and $\Delta\rho_{\min}^c$. The limiting factor is to limit $\Delta\rho_i^v$ to be within the range defined by $\Delta\rho_{\max}^c$ and $\Delta\rho_{\min}^c$. It can be expressed as

$$\phi_i = \begin{cases} \frac{\Delta\rho_{\max}^c}{\Delta\rho_i^v} & \text{if } \Delta\rho_i^v > \Delta\rho_{\max}^c \\ \frac{\Delta\rho_{\min}^c}{\Delta\rho_i^v} & \text{if } \Delta\rho_i^v < \Delta\rho_{\min}^c \\ 1 & \text{otherwise} \end{cases} \quad (19)$$

As can be seen, $0 < \phi_i \leq 1$. The final limiter for the cell is obtained by taking the minimum value of ϕ_i , i.e. $\phi = \min_i(\phi_i)$.

3. PARALLELIZATION AND VECTORIZATION

CaMEL_Aero was first parallelized on the Cray T3E-1200 using the ParMETIS mesh partitioning (Karypis and Kumar 1998) and the MPI parallel programming module. We have communications requirement for vertices, faces and cells on the partition boundaries. The inter-processor gather/scatter communications subroutines for faces and cells are written directly based on those for nodes which has been extensively used in our finite element solvers (Aliabadi and Tezduyar 1993; Aliabadi and Tezduyar 2000). Very efficient non-blocking (MPI) functions are called to set up the inter-processor “gather” and “scatter” routines in the pre-processing stage. The resultant parallel scalability performance is excellent.

CaMEL_Aero has also been fully vectorized (Tu, Aliabadi et al. 2005b) on the Cray X1 parallel/vector/multi-streaming architecture. Inside each multistreaming processor (MSP), the compiler will try to multi-stream and vectorize each loop. For the loop to be

successfully vectorized by the compiler, the loop must be free of any of the following: data dependencies, memory contention, I/O statements, non-inlined calls to subroutines and functions. The very useful compiler option “-rm” can be used to prompt the compiler to generate a *.lst* file for each source file. The *.lst* file reports the multi-streaming and vectorization status of each loop in the source file. If the loop is fully multi-streamed and vectorized, each line of the loop will be marked with “MV” at the beginning of that line.

In *CaMEL_Aero*, a face-based loop is used to compute the fluxes across each face. The result is then scattered to the two adjacent cells of the face. The “add” operation assembles the global cell-based vector composed of the residual. Therefore, the Cray X1E compiler will not vectorize loops like this by default because of these memory scatter statements, and if we force the compiler to vectorize the loop, it is possible that two faces access the same cell simultaneously causing memory contention. Another situation that prohibits vectorization is the matrix-free LU-SGS preconditioner. The LU-SGS preconditioner involves solving two block triangular linear equation systems which require sequential operations. Note that the face loop and the LU-SGS preconditioner are extensively called by the GMRES solver. These two situations consume the vast majority of the total computational time. Therefore, the vectorization of these two situations is crucial to achieve the optimal performance of the code on the Cray X1E.

3.1 Vectorization of face loops using face grouping

To vectorize the face loops, we divide the faces into groups. Inside each group, no two faces share the same adjacent cell. Thus, the memory contention problem can be avoided. The face grouping can also be designated as face-coloring scheme which is similar to the element-coloring scheme used in vectorizing the node-based finite element solvers (Johnson 2003; Aliabadi, Johnson et al. 2004). This grouping process is done in the pre-processing stage. With this algorithm, face groups are created to contain as many faces as possible. Once the faces are grouped, we will modify the face loop to contain an outer group loop and an inner face loop. Since we have guaranteed that there will be no memory contention inside each face group, we can force the vectorization of the inner face loop by applying the “CONCURRENT” compiler directive. The pseudo code of the new face loop is shown below:

```
DO IG = 1, NGF
! NGF is the number of face groups.

    IFACE_BEG = FGROUP(IG)
    IFACE_END = FGROUP(IG+1) - 1
```

```

!DIR$ CONCURRENT
DO IFACE = IFACE_BEG, IFACE_END
  IE1 = IFE(1,IFACE)
  IE2 = IFE(2,IFACE)
  ...
  ...! perform flux computations.
  ...
  ...! scattered to adjacent cells.
ENDDO
ENDDO

```

For pure tetrahedral meshes, the face coloring algorithm will divide all faces into about 7 groups on each processor. The left panel in Fig. 1 shows the grouping statistics on a typical processor for a tetrahedral mesh containing 3,652,436 cells and 7,347,956 faces and partitioned by 24 processors. The right panel in Fig. 1 shows the grouping statistics information on a typical processor about a pure hexahedral mesh containing 40,151,112 hexahedra and 120,604,576 faces. 64 processors are used to partition this mesh. On each processor the faces are divided into 8-9 groups. As can be seen from Fig. 1, on average, the majority of the groups contain a large number of faces with one or two smaller groups for the remainder of faces. The number of faces in each group determines the “vector length” for the face loops. With the full vectorization of the face loops, very high-sustained performance can be achieved.

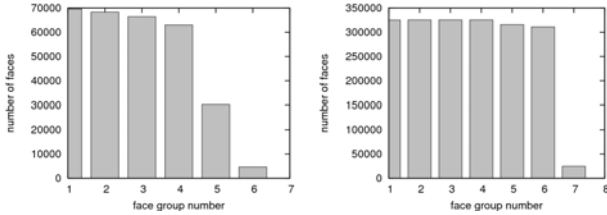


Fig. 1: Typical face grouping statistics. Left: a pure tetrahedral mesh with $ne = 3,652,436$ and $nproc = 24$. Right: a pure hexahedral mesh. $ne = 40,151,112$ and $nproc = 64$.

3.2 Vectorization of the LU-SGS preconditioner

To vectorize the LU-SGS preconditioner, we apply the truncated Neumann expansions of the inverse of triangular matrices (Benzi and Tuma 1999). For example, the Neumann expansion of the inverse of the lower triangular matrix is given by

$$(\mathbf{I} + \mathbf{L})^{-1} = \sum_{k=0}^{n-1} (-1)^k \mathbf{L}^k \quad (20)$$

With the truncated Neumann expansions, the sequential computations needed to solve the triangular system exactly can be reduced to a finite number of matrix-vector multiplications, i.e. the \mathbf{L} operator in Eq.

(20). To utilize Eq. (20), we must rewrite the preconditioning matrix given by Eq. (10) as:

$$\mathbf{P} = (\mathbf{I} + \hat{\mathbf{L}})\mathbf{D}(\mathbf{I} + \hat{\mathbf{U}}) \quad (21)$$

with $\hat{\mathbf{L}} = \mathbf{L}\mathbf{D}^{-1}$ and $\hat{\mathbf{U}} = \mathbf{D}^{-1}\mathbf{U}$.

Correspondingly, the forward sweep and the backward sweep become

$$(\mathbf{I} + \hat{\mathbf{L}})\mathbf{v}^* = \mathbf{v} \quad (22)$$

and

$$(\mathbf{I} + \hat{\mathbf{U}})\mathbf{v}^\# = \mathbf{D}^{-1}\mathbf{v}^* \quad (23)$$

respectively. Recall that the diagonal matrix \mathbf{D} is a scalar constant for each cell, thus producing no difficulty in the vectorization. If the first two terms of the Neumann expansion are kept, then \mathbf{v}^* can be computed via

$$\begin{aligned} \mathbf{v}^* &= \mathbf{v} - \hat{\mathbf{L}}\mathbf{v} \\ &= \mathbf{v} - \mathbf{L}(\mathbf{D}^{-1}\mathbf{v}) \end{aligned} \quad (24)$$

Similar expressions can be obtained for $\mathbf{v}^\#$. Obviously, the vectorization can be easily achieved using Eq. (24). Numerical experiments show that keeping the first two terms of the Neumann expansion is sufficient to yield satisfactory convergence. The outcome of the vectorization is that the solver is roughly 50 times faster on the Cray X1 than on the Cray T3E (Tu, Aliabadi et al. 2005b).

4. PERFORMANCE ANALYSIS

4.1 Slope limiter performance

Here we use a simple oblique shock reflection problem ($M_\infty = 2.9$, shock angle of 29°) to test the limiter performance. The computational domain is a 4×1 rectangle. Fig. 2(a-c) shows the pressure distribution at horizontal cut $y = 0.5$ compared with exact solution. Fig. 2(d) shows the convergence histories with different limiter parameter values. The effect of ε in Eq. (17) can be clearly seen. When $\varepsilon = 10^{-4}$, the overshoots/undershoots around the shocks have been completely suppressed while the convergence of residual is only slightly affected.

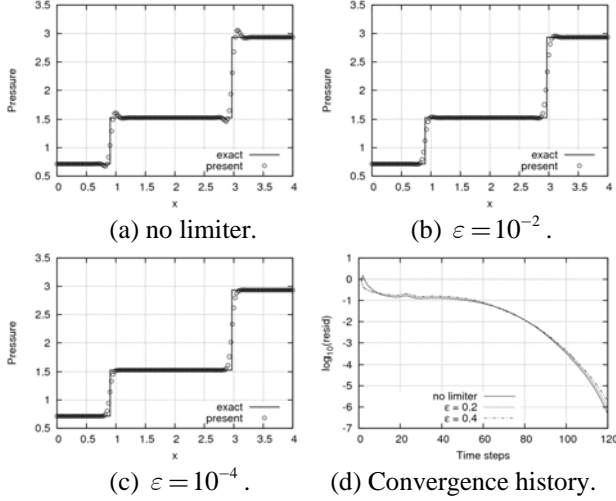


Fig. 2: Performance of the limiter. Pressure at $y = 0.5$ (a-c) and convergence history (d) for Shock reflect problem.

4.2 Performance for flows at various Mach numbers

We first present the solution of a low subsonic ($M_\infty = 0.1$) inviscid flow around a cylinder. At this Mach number, the flow can be assumed incompressible and analytical solution is available for the pressure and velocity distributions on the cylinder surface. As can be seen in Fig. 3, the pressure contours exhibit a nearly perfect symmetry that is true for inviscid incompressible flows. The pressure and velocity distributions agree very well with analytical solutions except that the vertical velocity component shows some discrepancy at the rear half of the cylinder surface.

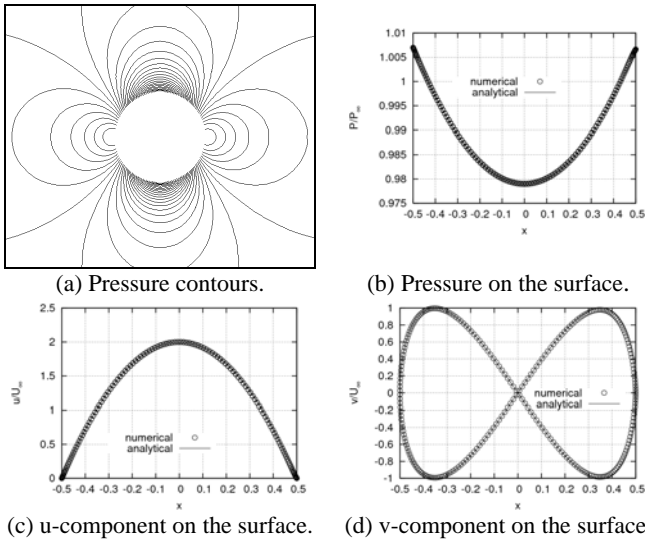


Fig. 3: Solutions of low subsonic ($M = 0.1$) flow around a circular cylinder.

Fig. 4 shows the solution of an inviscid hypersonic flow ($M_\infty = 15$) passing around a blunted body. To avoid the carbuncle problem, we choose the modified Steger-Warming scheme (Scalabrini and Boyd 2005) in simulating such supersonic blunted body problems. No carbuncle phenomenon can be seen in Fig. 4 and the predicted pressure jump across the normal shock agrees very well with the theoretical value.

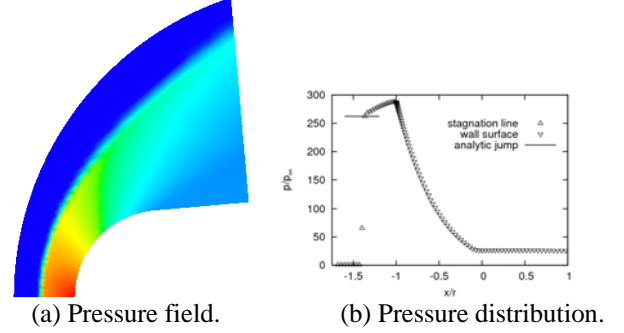


Fig. 4: Solution of hypersonic flow ($M = 15$) around a blunted body.

4.3 Parallel scalability

We run a test case of about 36 million hybrid prism/tetrahedron cells on the linux cluster JVN located in Army Research Laboratory (ARL). The case was run using 16, 32, 64, 128, 256 and 512 processors. Another case is a tetrahedral grid of 17 million elements. This case was run on Cray X1E using 8, 32, and 64 processors. The speedup vs. number of processors is shown in Fig. 5. As can be seen, *CaMEL_Aero* exhibits excellent scalability.

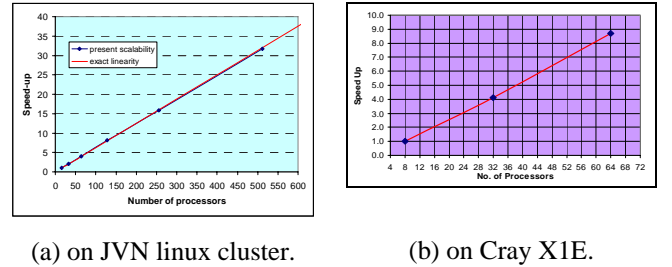


Fig. 5: Scalability performance of *CaMEL_Aero*.

4.4 Accuracy in predicting aerodynamic forces

The first case is a laminar subsonic flow at $M_\infty = 0.5$ and $Re = 5000$ around the NACA0012 airfoil. The computational mesh contains 15649 hybrid triangular/quadrilateral cells. Fig 6(a) and (b) are the drag and lift convergence histories, respectively. Fig 6(c) and (d) show the pressure and friction coefficients distribution on the airfoil surface. Table 1 lists the computed drag and lift coefficients with and without slope limiting. The results agree well with results in the literature. It can be

seen that slope limiting slightly increase the force coefficients but does not degrade the accuracy.

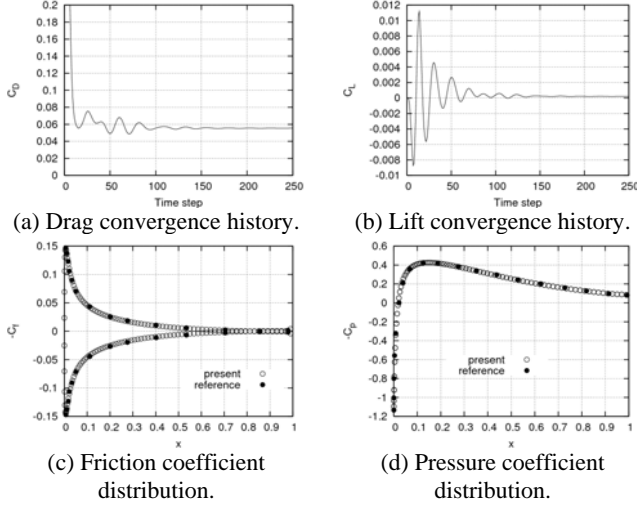


Fig 6: Solution of subsonic laminar flow ($M = 0.5$, $Re = 5000$, $AoA = 0^\circ$) around NACA0012 airfoil.

Table 1: Computed Force Coefficients of Subsonic Laminar Flow around the NACA0012 Airfoil.

	c_D		c_L	
	no limiter	limiter, $\varepsilon = 0.01$	no limiter	limiter, $\varepsilon = 0.01$
Due to pressure	0.02321007	0.02315961	0.00019712	0.00022851
Due to friction	0.03216580	0.03228044	0.00000164	0.00000183
Total	0.05537587	0.05544005	0.00019876	0.00023034

For turbulent flows, the Spalart-Allmaras Detached Eddy Simulation (SA-DES) turbulence model is employed to compute the eddy viscosity. The distance to the closest wall is computed with the help from KDTree2 (Kennel 2004), a freely available software to efficiently search the nearest neighbors. We follow the guideline in (Lin, Percival et al. 1995) to generate the high aspect ratio cells near the body. The first layer thickness is a function of the Reynolds number, i.e.

$$y^+ = 0.172 y^* Re^{0.9} \quad (25)$$

where y^* is the non-dimensional first layer thickness next to the body. $y^+ \approx 1$ is assumed in the first layer during the mesh generation stage.

Fig. 7 shows the eddy viscosity field of a low subsonic flow passing around the NACA0015 airfoil. The flight conditions are $M_\infty = 0.1235$, $Re = 1.5 \times 10^6$ and $AoA = 12^\circ$. The eddy viscosity at the rear part of the top

surface of the airfoil is essential for the correct prediction of the aerodynamic drag and lift.

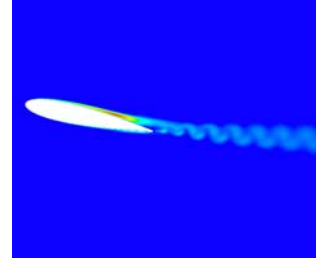


Fig. 7: Eddy viscosity field of a low subsonic flow around NACA0015 airfoil ($M_\infty = 0.1235$, $Re = 1.5 \times 10^6$).

We also run a simulation about a supersonic flow passing around a spinning bullet. The flight conditions are $M_\infty = 2.7$ and $Re = 9.71 \times 10^5$. The mesh contains about 40,151,112 unstructured hexahedral cells. Fig. 8 shows the Mach number field. The base area is highly turbulent. The computed drag agrees favorably with the experimental data of 0.279.

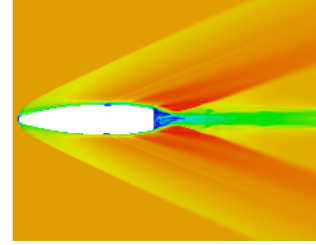


Fig. 8: Mach number field of supersonic flow around a spinning bullet ($M_\infty = 2.7$, $Re = 9.71 \times 10^5$).

4.5 Speed analysis

The speed of the *CaMEL_Aero* is measured using a time-scale T_c defined as:

$$T_c = \frac{n_{proc}}{n_{elem} n_{ts} n_{it} n_k} T_{run} \quad (26)$$

where n_{proc} , n_{elem} , n_{ts} , n_{it} , and n_k are the numbers of processors, elements, time steps, nonlinear iterations within each timestep, size of Krylov space, respectively, and T_{run} is the CPU time. For the *CaMEL_Aero* flow solver, the typical speed for an all-tet mesh simulation is about 1.0E-6 second on the Cray X1E for solving Navier-Stokes equation and 3.4E-7 second for the turbulence equation. The speed for all-hex meshes is about 1.3 times that of all-tet meshes. The speed of solving turbulence equations is roughly one third of that of solving the Navier-Stokes equations, though the SA-DES turbulence equation contains only one unknown while the Navier-

Stokes equations contain 5 unknowns. This can be attributed to the vast indirect addressing using unstructured data structures.

4.6 Memory requirement

The base memory (excluding krylov vectors) requirement is between 1.15-1.41 kbytes/element (considering 1 mbytes = 1048576 byte) depending on the mesh types; obviously, an all-hex mesh consumes more memory than an all-tet mesh on the element basis. The Krylov vector memory use is exactly 40 bytes/element/krylov (since we have 5 unknowns each element), regardless of element type. Since DES is decoupled from the Navier-Stokes equations, the Krylov vector space can be reused for the turbulence equation.

CONCLUSIONS

In this paper, we first describe some key ingredients in developing *CaMEL_Aero*, an unstructured finite volume solver for compressible flows. The focus is put on the Jacobian-free GMRES solver and the matrix-free LU-SGS preconditioning technique. We also present our implementation of a simple slope limiting procedure which is suitable for arbitrarily unstructured meshes. We then discuss the parallelization and vectorization of the solver on parallel and vector computer platforms. The emphasis is put on the vectorization of face loops and the LU-SGS preconditioner. Finally, we analyze the performance of *CaMEL_Aero* through a few numerical examples. The performance shows that *CaMEL_Aero* is an accurate, efficient and robust numerical tool in compressible flow simulations.

ACKNOWLEDGMENTS

This work is funded by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory contract numbers DAAD19-01-2-0014 and DAAD19-03-D-0001.

REFERENCES

Aliabadi, S., Johnson, A. et al., 2004: Simulation of contaminant dispersion on the Cray X1: verification and implementation. *Journal of Aerospace Computing, Information and Communication* **1**(8): 341-361.

Aliabadi, S. and Tezduyar, T., 1993: Space-time finite element computation of compressible flows involving moving boundaries and interfaces." *Computer Methods in Applied Mechanics and Engineering* **107**: 209-223.

Aliabadi, S. and Tezduyar, T., 2000: Stabilized-finite-element/interface-capturing technique for parallel computation of unsteady flows with interfaces. *Computer*

Methods for Applied Mechanics and Engineering **190**: 243-261.

Benzi, M. and Tuma, M., 1999: A comparative study of sparse approximate inverse preconditioner. *Applied Numerical Mathematics* **30**: 305-340.

Harten, A., Lax, P. et al., 1983: On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review* **25**: 35-61.

Johnson, A., 2003: Computational fluid dynamics applications on the Cray X1 architecture: experiences, algorithms, and performance analysis. *Cray User Group Conference 2003 Proceedings*.

Karypis, G. and Kumar, V., 1998: Metis 4.0: unstructured graph partitioning and sparse matrix ordering systems. Technical report, University of Minnesota.

Kennel, M. B., 2004: KDTREE2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space, http://arxiv.org/PS_cache/physics/pdf/0408/0408067.pdf.

Knoll, D. and Keyes, D., 2004: Jacobian-free Newton-Krylov methods: a survey of approaches and applications. *Journal of Computational Physics* **193**: 357-397.

Lin, C.-W., Percival, S. et al., 1995: Viscous drag calculations for ship hull geometry. Technical report. Bethesda, Mariland, Carderock Division Naval Surface Warfare Center.

Luo, H., Baum, J. et al., 1998: A fast matrix-free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics* **146**: 664-690.

Mathur, S. R. and Murphy, J. Y., 1997: A pressure-based method for unstructured meshes. *Numerical Heat Transfer, Part B* **31**: 195-215.

Nikitin, N. V., Nicoud, F. et al., 2000: An approach to wall modeling in large-eddy simulations. *Physics of Fluids* **12**(7): 1629-1632.

Saad, Y., 1996: *Iterative methods for sparse linear systems*, PWS Publishing Company.

Scalabrin, L. C. and Boyd, I. D., 2005: Development of an unstructured Navier-Stokes solver for hyperbolic nonequilibrium aerothermodynamics. *38th AIAA Thermophysics Conference*. Canada, AIAA paper 2005-5203.

Sharov, D., Luo, H. et al., 2000: Implementation of unstructured grid GMRES+LU-SGS method on shared-memory cache-based parallel computers, AIAA paper 2000-0927.

Toro, E., 1999: *Riemann solvers and numerical methods for fluid dynamics*. New York, Springer.

Tu, S. and Aliabadi, S., 2005a: A slope limiting procedure in discontinuous Galerkin finite element method for gasdynamics applications. *International Journal of Numerical Analysis and Modeling* **2**(2): 163-178.

Tu, S., Aliabadi, S. et al., 2005b: High performance computation of compressible flows on the Cray X1. *Proceedings of the Second International Conference on Computational Ballistics*, Cordoba, Spain.

Tu, S., Aliabadi, S. et al., 2005c: A robust parallel implicit finite volume solver for high-speed compressible flows. *43rd AIAA Aerospace Sciences Meeting and Exhibit*. Reno, Nevada, AIAA paper 2005-1396.